

TensorFlow – 相关 API

TensorFlow 相关函数理解

任务时间：时间未知

tf.nn.conv2d

```
conv2d(  
    input,  
    filter,  
    strides,  
    padding,  
    use_cudnn_on_gpu=True,  
    data_format='NHWC',  
    name=None  
)
```

功能说明：

卷积的原理可参考 *A guide to convolution arithmetic for deep learning*

参数列表：

参数名	必选	类型	说明
input	是	tensor	是一个 4 维的 tensor，即 [batch, in_height, in_width, in_channels]（若 input 是图像，[训练时一个 batch 的图片数量, 图片高度, 图片宽度, 图像通道数]）
filter	是	tensor	是一个 4 维的 tensor，即 [filter_height, filter_width, in_channels, out_channels]（若 input 是图像，[卷积核的高度, 卷积核的宽度, 图像通道数, 卷积核个数]），filter 的 in_channels 必须和 input 的 in_channels 相等
strides	是	列表	长度为 4 的 list，卷积时候在 input 上每一维的步长，一般 strides[0] = strides[3] = 1
padding	是	string	只能为 " VALID ", " SAME " 中之一，这个值决定了不同的卷积方式。VALID 丢弃方式；SAME：补全方式

参数名	必选	类型	说明
use_cudnn_on_gpu	否	bool	是否使用 cudnn 加速, 默认为 true
data_format	否	string	只能是 " NHWC ", " NCHW ", 默认 " NHWC "
name	否	string	运算名称

Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

0	0	0	0	0	0	0
0	2	2	1	2	2	0
0	1	2	0	2	0	0

0	2	0	0	2	1	0
0	0	0	1	2	1	0
0	0	2	2	0	2	0
0	0	0	0	0	0	0

 $x[:, :, 1]$

0	0	0	0	0	0	0
0	0	2	0	0	2	0
0	1	2	0	2	1	0

0	1	2	0	1	2	0
0	1	0	2	2	2	0
0	0	1	1	0	1	0
0	0	0	0	0	0	0

 $x[:, :, 2]$

0	0	0	0	0	0	0
0	1	1	2	1	0	0
0	0	2	1	0	2	0

0	2	1	2	2	2	0
0	2	1	2	0	2	0
0	1	0	2	2	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

 $w0[:, :, 0]$

-1	1	1
-1	1	-1
1	-1	1

 $w0[:, :, 1]$

1	-1	-1
-1	0	-1
-1	0	1

 $w0[:, :, 2]$

1	-1	-1
-1	1	0
-1	1	1

Bias b0 (1x1x1)

 $b0[:, :, 0]$

1

Filter W1 (3x3x3)

 $w1[:, :, 0]$

-1	-1	1
-1	1	-1
1	1	0

 $w1[:, :, 1]$

0	0	1
-1	0	0
1	0	-1

 $w1[:, :, 2]$

-1	1	0
-1	0	-1
0	-1	0

Bias b1 (1x1x1)

 $b1[:, :, 0]$

0

Output Volume

 $o[:, :, 0]$

3	-4	2
0	-2	-7
-7	-4	-2

 $o[:, :, 1]$

-2	-6	1
2	-9	-1
0	1	-1

toggle movement

示例代码:

现在您可以在 `/home/ubuntu` 目录下创建源文件 `conv2d.py`, 内容可参考:

示例代码: `/home/ubuntu/conv2d.py`

```
import tensorflow as tf

a = tf.constant([1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,0,0,1,1,0,0],dtype=tf.float32,shape=[1,5,5,1])
b = tf.constant([1,0,1,0,1,0,1,0,1],dtype=tf.float32,shape=[3,3,1,1])
c = tf.nn.conv2d(a,b,strides=[1, 2, 2, 1],padding='VALID')
d = tf.nn.conv2d(a,b,strides=[1, 2, 2, 1],padding='SAME')
with tf.Session() as sess:
    print ("c shape:")
    print (c.shape)
    print ("c value:")
    print (sess.run(c))
    print ("d shape:")
    print (d.shape)
    print ("d value:")
    print (sess.run(d))
```

然后执行:

```
cd /home/ubuntu;
python conv2d.py
```

执行结果:

```
c shape:
(1, 3, 3, 1)
c value:
[[[ 4.]
  [ 3.]
  [ 4.]]

 [[ 2.]
  [ 4.]
  [ 3.]]

 [[ 2.]
  [ 3.]
```

```

    [ 4.]]]]
d shape:
(1, 5, 5, 1)
d value:
[[[ [ 2.]
      [ 2.]
      [ 3.]
      [ 1.]
      [ 1.]]

  [[ 1.]
   [ 4.]
   [ 3.]
   [ 4.]
   [ 1.]]

  [[ 1.]
   [ 2.]
   [ 4.]
   [ 3.]
   [ 3.]]

  [[ 1.]
   [ 2.]
   [ 3.]
   [ 4.]
   [ 1.]]

  [[ 0.]
   [ 2.]
   [ 2.]
   [ 1.]
   [ 1.]]]]

```

tf.nn.relu

```

relu(
    features,
    name=None
)

```

功能说明:

relu激活函数可以参考 *CS231n: Convolutional Neural Networks for Visual Recognition*

参数列表：

参数名	必选	类型	说明
features	是	tensor	是以下类型float32, float64, int32, int64, uint8, int16, int8, uint16, half
name	否	string	运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `relu.py`，内容可参考：

示例代码： `/home/ubuntu/relu.py`

```
import tensorflow as tf

a = tf.constant([1,-2,0,4,-5,6])
b = tf.nn.relu(a)
with tf.Session() as sess:
    print (sess.run(b))
```

然后执行：

```
cd /home/ubuntu;
python relu.py
```

执行结果：

```
[1 0 0 4 0 6]
```

tf.nn.max_pool

```
max_pool(
    value,
    ksize,
    strides,
    padding,
    data_format='NHWC',
```

```
    name=None
)
```

功能说明：

池化的原理可参考 *CS231n: Convolutional Neural Networks for Visual Recognition*

参数列表：

参数名	必选	类型	说明
value	是	tensor	4 维的张量，即 [batch, height, width, channels]，数据类型为 tf.float32
ksize	是	列表	池化窗口的大小，长度为 4 的 list，一般是 [1, height, width, 1]，因为不在 batch 和 channels 上做池化，所以第一个和最后一个维度为 1
strides	是	列表	池化窗口在每一个维度上的步长，一般 strides[0] = strides[3] = 1
padding	是	string	只能为 " VALID "，" SAME " 中之一，这个值决定了不同的池化方式。VALID 丢弃方式；SAME：补全方式
data_format	否	string	只能是 " NHWC "，" NCHW "，默认" NHWC "
name	否	string	运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `max_pool.py`，内容可参考：

示例代码： `/home/ubuntu/max_pool.py`

```
import tensorflow as tf

a = tf.constant([1,3,2,1,2,9,1,1,1,3,2,3,5,6,1,2],dtype=tf.float32,shape=[1,4,4,1])
b = tf.nn.max_pool(a,ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1],padding='VALID')
c = tf.nn.max_pool(a,ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1],padding='SAME')
with tf.Session() as sess:
    print ("b shape:")
    print (b.shape)
    print ("b value:")
    print (sess.run(b))
```

```
print ("c shape:")
print (c.shape)
print ("c value:")
print (sess.run(c))
```

然后执行：

```
cd /home/ubuntu;
python max_pool.py
```

执行结果：

```
b shape:
(1, 2, 2, 1)
b value:
[[[ 9.]
  [ 2.]]

 [[ 6.]
  [ 3.]]]]
c shape:
(1, 2, 2, 1)
c value:
[[[ 9.]
  [ 2.]]

 [[ 6.]
  [ 3.]]]]
```

tf.nn.dropout

```
dropout(
    x,
    keep_prob,
    noise_shape=None,
    seed=None,
    name=None
)
```

功能说明：

原理可参考 *CS231n: Convolutional Neural Networks for Visual Recognition*

参数列表：

参数名	必选	类型	说明
x	是	tensor	输出元素是 x 中的元素以 keep_prob 概率除以 keep_prob，否则为 0
keep_prob	是	scalar Tensor	dropout 的概率，一般是占位符
noise_shape	否	tensor	默认情况下，每个元素是否 dropout 是相互独立。如果指定 noise_shape，若 noise_shape[i] == shape(x)[i]，该维度的元素是否 dropout 是相互独立，若 noise_shape[i] != shape(x)[i] 该维度元素是否 dropout 不相互独立，要么一起 dropout 要么一起保留
seed	否	数值	如果指定该值，每次 dropout 结果相同
name	否	string	运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `dropout.py`，内容可参考：

示例代码： `/home/ubuntu/dropout.py`

```
import tensorflow as tf

a = tf.constant([1,2,3,4,5,6],shape=[2,3],dtype=tf.float32)
b = tf.placeholder(tf.float32)
c = tf.nn.dropout(a,b,[2,1],1)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print (sess.run(c,feed_dict={b:0.75}))
```

然后执行：

```
cd /home/ubuntu;
python dropout.py
```

执行结果：

```
[[ 0.          0.          0.          ]
 [ 5.33333349  6.66666651  8.          ]]
```

tf.nn.sigmoid_cross_entropy_with_logits

```
sigmoid_cross_entropy_with_logits(  
    _sentinel=None,  
    labels=None,  
    logits=None,  
    name=None  
)
```

功能说明：

先对 logits 通过 sigmoid 计算，再计算交叉熵，交叉熵代价函数可以参考 *CS231n: Convolutional Neural Networks for Visual Recognition*

参数列表：

参数名	必选	类型	说明
_sentinel	否	None	没有使用的参数
labels	否	Tensor type, shape 与 logits相同	
logits	否	Tensor type 是 float32 或者 float64	
name	否	string	运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `sigmoid_cross_entropy_with_logits.py`:

示例代码： `/home/ubuntu/sigmoid_cross_entropy_with_logits.py`

```
import tensorflow as tf  
x = tf.constant([1,2,3,4,5,6,7],dtype=tf.float64)  
y = tf.constant([1,1,1,0,0,1,0],dtype=tf.float64)  
loss = tf.nn.sigmoid_cross_entropy_with_logits(labels = y,logits = x)  
with tf.Session() as sess:  
    print (sess.run(loss))
```

然后执行：

```
cd /home/ubuntu;
python sigmoid_cross_entropy_with_logits.py
```

执行结果：

```
[ 3.13261688e-01  1.26928011e-01  4.85873516e-02  4.01814993e+00
 5.00671535e+00  2.47568514e-03  7.00091147e+00]
```

tf.truncated_normal

```
truncated_normal(
    shape,
    mean=0.0,
    stddev=1.0,
    dtype=tf.float32,
    seed=None,
    name=None
)
```

功能说明：

产生截断正态分布随机数，取值范围为 $[\text{mean} - 2 * \text{stddev}, \text{mean} + 2 * \text{stddev}]$ 。

参数列表：

参数名	必选	类型	说明
shape	是	1 维整形张量或 array	输出张量的维度
mean	否	0 维张量或数值	均值
stddev	否	0 维张量或数值	标准差
dtype	否	dtype	输出类型
seed	否	数值	随机种子，若 seed 赋值，每次产生相同随机数
name	否	string	运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `truncated_normal.py`:

示例代码: `/home/ubuntu/truncated_normal.py`

```
import tensorflow as tf
initial = tf.truncated_normal(shape=[3,3], mean=0, stddev=1)
print(tf.Session().run(initial))
```

然后执行:

```
python /home/ubuntu/truncated_normal.py
```

执行结果:

将得到一个取值范围 `[-2, 2]` 的 `3 * 3` 矩阵, 您也可以尝试修改源代码看看输出结果有什么变化?

tf.constant

```
constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
    verify_shape=False
)
```

功能说明:

根据 `value` 的值生成一个 `shape` 维度的常量张量

参数列表:

参数名	必选	类型	说明
<code>value</code>	是	常量数值或者 list	输出张量的值

参数名	必选	类型	说明
dtype	否	dtype	输出张量元素类型
shape	否	1 维整形张量或 array	输出张量的维度
name	否	string	张量名称
verify_shape	否	Boolean	检测 shape 是否和 value 的 shape 一致，若为 False，不一致时，会用最后一个元素将 shape 补全

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `constant.py`，内容可参考：

示例代码： `/home/ubuntu/constant.py`

```
#!/usr/bin/python

import tensorflow as tf
import numpy as np
a = tf.constant([1,2,3,4,5,6],shape=[2,3])
b = tf.constant(-1,shape=[3,2])
c = tf.matmul(a,b)

e = tf.constant(np.arange(1,13,dtype=np.int32),shape=[2,2,3])
f = tf.constant(np.arange(13,25,dtype=np.int32),shape=[2,3,2])
g = tf.matmul(e,f)
with tf.Session() as sess:
    print (sess.run(a))
    print ("#####")
    print (sess.run(b))
    print ("#####")
    print (sess.run(c))
    print ("#####")
    print (sess.run(e))
    print ("#####")
    print (sess.run(f))
    print ("#####")
    print (sess.run(g))
```

然后执行：

```
python /home/ubuntu/constant.py
```

执行结果：

a: 2x3 维张量;
b: 3x2 维张量;
c: 2x2 维张量;
e: 2x2x3 维张量;
f: 2x3x2 维张量;
g: 2x2x2 维张量。

tf.placeholder

```
placeholder(  
    dtype,  
    shape=None,  
    name=None  
)
```

功能说明：

是一种占位符，在执行时候需要为其提供数据

参数列表：

参数名	必选	类型	说明
dtype	是	dtype	占位符数据类型
shape	否	1 维整形张量或 array	占位符维度
name	否	string	占位符名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `placeholder.py`，内容可参考：

示例代码：`/home/ubuntu/placeholder.py`

```
#!/usr/bin/python

import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32,[None,3])
y = tf.matmul(x,x)
with tf.Session() as sess:
    rand_array = np.random.rand(3,3)
    print(sess.run(y,feed_dict={x:rand_array}))
```

然后执行:

```
python /home/ubuntu/placeholder.py
```

执行结果:

输出一个 3x3 的张量

tf.nn.bias_add

```
bias_add(
    value,
    bias,
    data_format=None,
    name=None
)
```

功能说明:

将偏差项 bias 加到 value 上面,可以看做是 tf.add 的一个特例,其中 bias 必须是一维的,并且维度和 value 的最后一维相同,数据类型必须和 value 相同。

参数列表:

参数名	必选	类型	说明
value	是	张量	数据类型为 float, double, int64, int32, uint8, int16, int8, complex64, or complex128
bias	是	1 维张量	维度必须和 value 最后一维维度相等

参数名	必选	类型	说明
data_format	否	string	数据格式, 支持 ' NHWC ' 和 ' NCHW '
name	否	string	运算名称

示例代码:

现在您可以在 `/home/ubuntu` 目录下创建源文件 `bias_add.py`, 内容可参考:

示例代码: `/home/ubuntu/bias_add.py`

```
#!/usr/bin/python

import tensorflow as tf
import numpy as np

a = tf.constant([[1.0, 2.0],[1.0, 2.0],[1.0, 2.0]])
b = tf.constant([2.0,1.0])
c = tf.constant([1.0])
sess = tf.Session()
print (sess.run(tf.nn.bias_add(a, b)))
#print (sess.run(tf.nn.bias_add(a,c))) error
print ("#####")
print (sess.run(tf.add(a, b)))
print ("#####")
print (sess.run(tf.add(a, c)))
```

然后执行:

```
python /home/ubuntu/bias_add.py
```

执行结果:

3 个 3x2 维张量。您也可以尝试修改源代码看看输出结果有什么变化?

tf.reduce_mean

```
reduce_mean(
    input_tensor,
```



```
axis=None,
keep_dims=False,
name=None,
reduction_indices=None
)
```

功能说明：

计算张量 input_tensor 平均值

参数列表：

参数名	必选	类型	说明
input_tensor	是	张量	输入待求平均值的张量
axis	否	None、0、1	None：全局求平均值；0：求每一列平均值；1：求每一行平均值
keep_dims	否	Boolean	保留原来的维度(例如不会从二维矩阵降为一维向量)
name	否	string	运算名称
reduction_indices	否	None	和 axis 等价，被弃用

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `reduce_mean.py`，内容可参考：

示例代码： `/home/ubuntu/reduce_mean.py`

```
#!/usr/bin/python

import tensorflow as tf
import numpy as np

initial = [[1.,1.],[2.,2.]]
x = tf.Variable(initial,dtype=tf.float32)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run(tf.reduce_mean(x)))
```

```
print(sess.run(tf.reduce_mean(x,0))) #Column  
print(sess.run(tf.reduce_mean(x,1))) #row
```

然后执行:

```
python /home/ubuntu/reduce_mean.py
```

执行结果:

```
1.5  
[ 1.5  1.5]  
[ 1.   2.]
```

tf.squared_difference

```
squared_difference(  
    x,  
    y,  
    name=None  
)
```

功能说明:

计算张量 x、y 对应元素差平方

参数列表:

参数名 必选 类型

说明

x 是 张量 是 half, float32, float64, int32, int64, complex64, complex128 其中一种类型

y 是 张量 是 half, float32, float64, int32, int64, complex64, complex128 其中一种类型

name 否 string 运算名称

示例代码:

现在您可以在 `/home/ubuntu` 目录下创建源文件 `squared_difference.py`, 内容可参考:

示例代码: /home/ubuntu/squared_difference.py

```
#!/usr/bin/python

import tensorflow as tf
import numpy as np

initial_x = [[1.,1.],[2.,2.]]
x = tf.Variable(initial_x,dtype=tf.float32)
initial_y = [[3.,3.],[4.,4.]]
y = tf.Variable(initial_y,dtype=tf.float32)
diff = tf.squared_difference(x,y)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run(diff))
```

然后执行:

```
python /home/ubuntu/squared_difference.py
```

执行结果:

```
[[ 4.  4.]
 [ 4.  4.]]
```

tf.square

```
square(
    x,
    name=None
)
```

功能说明:

计算张量对应元素平方

参数列表:

参数名 必选 类型

说明

x 是 张量 是 half, float32, float64, int32, int64, complex64, complex128 其中一种类型
name 否 string 运算名称

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `square.py`，内容可参考：

示例代码： `/home/ubuntu/square.py`

```
#!/usr/bin/python
import tensorflow as tf
import numpy as np

initial_x = [[1.,1.],[2.,2.]]
x = tf.Variable(initial_x,dtype=tf.float32)
x2 = tf.square(x)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print(sess.run(x2))
```

然后执行:

```
python /home/ubuntu/square.py
```

执行结果：

```
[[ 1.  1.]
 [ 4.  4.]]
```

TensorFlow 相关类理解

任务时间：时间未知

tf.Variable

```

__init__(
    initial_value=None,
    trainable=True,
    collections=None,
    validate_shape=True,
    caching_device=None,
    name=None,
    variable_def=None,
    dtype=None,
    expected_shape=None,
    import_scope=None
)

```

功能说明：

维护图在执行过程中的状态信息，例如神经网络权重值的变化。

参数列表：

参数名	类型	说明
initial_value	张量	Variable 类的初始值，这个变量必须指定 shape 信息，否则后面 validate_shape 需设为 False
trainable	Boolean	是否把变量添加到 collection GraphKeys.TRAINABLE_VARIABLES 中（collection 是一种全局存储，不受变量名生存空间影响，一处保存，到处可取）
collections	Graph collections	全局存储，默认是 GraphKeys.GLOBAL_VARIABLES
validate_shape	Boolean	是否允许被未知维度的 initial_value 初始化
caching_device	string	指明哪个 device 用来缓存变量
name	string	变量名
dtype	dtype	如果被设置，初始化的值就会按照这个类型初始化
expected_shape	TensorShape	要是设置了，那么初始的值会是这种维度

示例代码：

现在您可以在 `/home/ubuntu` 目录下创建源文件 `Variable.py`，内容可参考：

示例代码: /home/ubuntu/Variable.py

```
#!/usr/bin/python

import tensorflow as tf
initial = tf.truncated_normal(shape=[10,10],mean=0,stddev=1)
W=tf.Variable(initial)
list = [[1.,1.],[2.,2.]]
X = tf.Variable(list,dtype=tf.float32)
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    print ("#####(1)#####")
    print (sess.run(W))
    print ("#####(2)#####")
    print (sess.run(W[:2,:2]))
    op = W[:2,:2].assign(22.*tf.ones((2,2)))
    print ("#####(3)#####")
    print (sess.run(op))
    print ("#####(4)#####")
    print (W.eval(sess)) #computes and returns the value of this variable
    print ("#####(5)#####")
    print (W.eval()) #Usage with the default session
    print ("#####(6)#####")
    print (W.dtype)
    print (sess.run(W.initial_value))
    print (sess.run(W.op))
    print (W.shape)
    print ("#####(7)#####")
    print (sess.run(X))
```

然后执行:

```
python /home/ubuntu/Variable.py
```

完成实验

任务时间: 时间未知

实验内容已完成

您可进行更多关于机器学习教程：

- [实验列表 – 机器学习](#)

关于 TensorFlow 的更多资料可参考 [TensorFlow 官网](#)。